Report Number: WUCSE-2004-84

2004-09-05

Intuitive tools for camera manipulation

Nisha Sudarsanam, Cindy Grimm, and Karan Singh

We present an image-space camera manipulation widget that sup-ports visualization of the relationship of the camera with respect tothe scene. The form of the widget presents the user with naturalaffordances for camera manipulation. Visual aids such as ghostingof the scene and preview animations are used to acquaint noviceusers with the functions of different parts of the widget. Mousegestures are used to transition between different perspective viewsof the scene in an intuitive way. Finally, we provide a novel methodfor visualizing camera bookmarks.

2004-84

Intuitive tools for camera manipulation

Authors: Sudarsanam, Nisha;Grimm, Cindy M.;Singh, Karan

September 5, 2004

Abstract: We present an image-space camera manipulation widget that supports visualization of the relationship of the camera with respect to the scene. The form of the widget presents the user with natural affordances for camera manipulation. Visual aids such as ghosting of the scene and preview animations are used to acquaint novice users with the functions of different parts of the widget. Mouse gestures are used to transition between different perspective views of the scene in an intuitive way. Finally, we provide a novel method for visualizing camera bookmarks.

Intuitive tools for camera manipulation

Category: Research

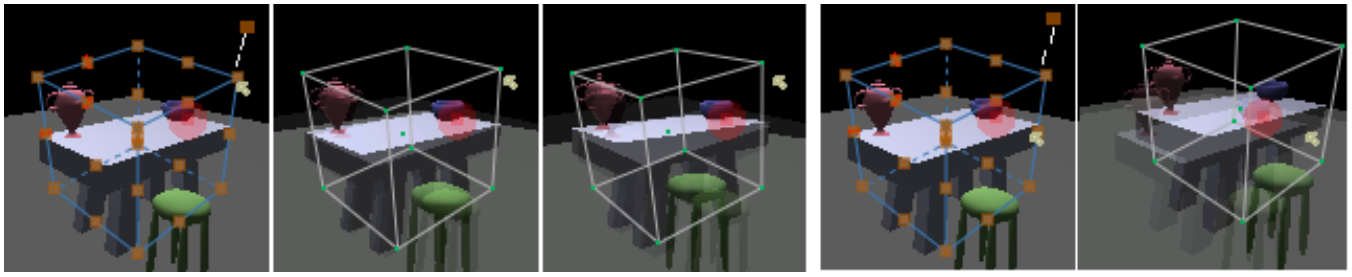


Figure 1: **Visual help:** The user hovers over a handle on the widget. The widget, along with the scene, animate to demonstrate how changing the handle changes the tool. Left: Rotation. Right: Pan.

Abstract

We present an image-space camera manipulation widget that supports visualization of the relationship of the camera with respect to the scene. The form of the widget presents the user with natural affordances for camera manipulation. Visual aids such as ghosting of the scene and preview animations are used to acquaint novice users with the functions of different parts of the widget. Mouse gestures are used to transition between different perspective views of the scene in an intuitive way. Finally, we provide a novel method for visualizing camera bookmarks.

Keywords: Camera control, Projection, Perspective

1 Introduction

Artists use the idea of linear perspective to translate the three dimensional world around them onto the two-dimensional surface of a painting. Objects in the picture are systematically foreshortened as they recede into the distance and orthogonal lines converge to one or more vanishing points. Note that the convergence of the lines depends entirely on the relative position of the camera to the viewed scene. When creating a projection, the artist often simplifies the geometry of the scene into sets of simple lines that represent these vanishing points. We propose that the visualization of these geometric relationships results in a more intuitive camera manipulation paradigm.

Most graphics packages treat the camera as a first-order object in the scene. The camera is treated as a separate, independent object in the scene with its own position and orientation. Camera manipulation consists of placing the camera in the scene, either using a "through-the-lens" metaphor or by manipulating a pictorial representation of the camera in another window.

In contrast, our system explicitly visualizes the camera-scene relationship in the image plane. The user then manipulates this relationship to change the camera projection. This allows the camera operations to be defined with respect to any point in the scene, and also provides information about how a particular camera manipulation will affect the current projection.

Artists often simplify complex scenes, representing them as lines, points and curves, because these are easier to visualize. Our widget mirrors this more intuitive approach by presenting the user with simplified geometric proxies of objects in the scene. The user then manipulates these geometric proxies to change the object's projection.

A full camera has 11 degrees of freedom, six for positioning and orienting the camera and 5 for controlling the projection [Michener and Carlbom 1980]. When manipulating a geometric proxy, not all 11 parameters of the camera have to be changed at once, nor do all desired projection changes map to a single parameter. For example, if the user moves the camera along the optical axis the perspective distortion changes, but the portion of the scene that is visible also changes. Therefore, it makes sense to change the view angle simultaneously to keep part of the scene the same size (see Figure 3).

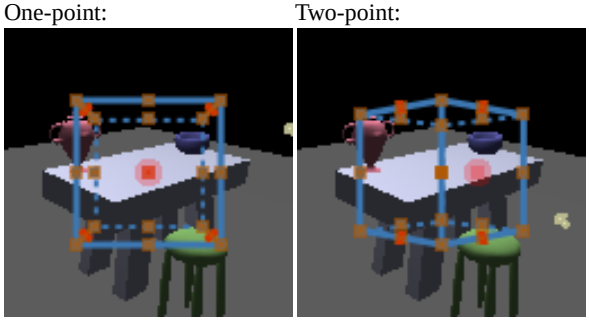
To acquaint novice users with the functions of the various handles on the widget, we use "just in time" animations. These animations act as preview animations indicating the possible changes that can be made to a camera if a particular handle is selected (see Figure 1).

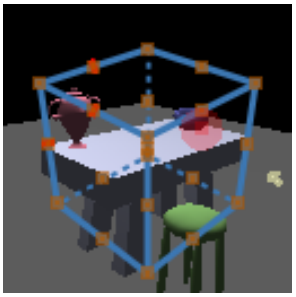
Finally, our widget allows users to save cameras and switch back to them at a later time. We refer to these saved cameras as "bookmarks". We provide two methods for visualizing the position of these camera bookmarks relative to the current camera in the 2D image plane.

1.1 Vision Of Approach

In this paper we specify a method of visualizing both the relationship of the camera with objects in the scene and the state of the camera itself. Using our method we can perform both these functions while staying in the 2D image plane where all the camera manipulations take place. To accomplish this we visualize the camera using the projection of a cube in the scene. The cube is a geometric proxy for some or all parts of the scene, and is a familiar piece of

One-point: Two-point:





Three-point:

Figure 2: **Vanishing points:** One, two and three-point perspective. The number of vanishing points depends on the orientation of the cube relative to the camera.

geometry that can be used to visualize changes in projection easily.

The cube can be viewed from three different perspective views (Figure 2). Each perspective view presents the user with a different subset of the changeable camera parameters that correspond naturally to how that perspective view can be changed. To further clarify what different handles on the widget do, we provide the user with visual feedback indicating how the scene changes when the widget changed. This visual feedback takes the form of ghosting of the scene with the new camera (see Figure 1). Preview animations of the several changes that the cube would go through from the current camera to the final camera also gives the user an idea of the changes that would be made to the camera without actually changing the current camera.

To make manipulation between perspective views of the cube simple we use a mouse gesture that allows the user to "spin" the 3D cube from one view to the next. This is also cued visually using ghosting.

Finally, we extend the idea of visualizing a single camera in the image plane to visualizing multiple, bookmarked cameras.

1.2 Contributions

This paper presents an intuitive widget that allows visualization of the relationship of the camera with the scene and the state of the camera, entirely in the 2D image plane. Visual aids such as ghosting of the scene and preview animations are used to acquaint the user with the functions of different parts of the widget. Gestures are used to transition between different perspective views of the scene in an intuitive way. Finally, we provide a novel way to visualize camera bookmarks.

1.3 Overview

We cover previous work in Section 2. Section 3 describes the widget itself, and how the user interacts with it. Section 4 describes the visual aids available for novice users. Finally, Section 5 describes bookmarks.

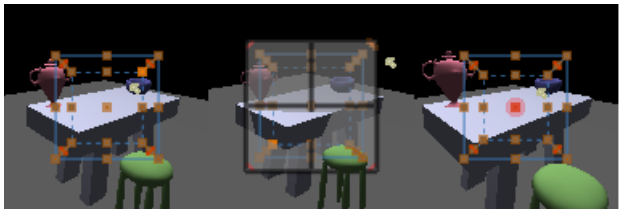


Figure 3: **Camera Dolly And Zoom With Focal Planes:** A camera dolly and zoom is accomplished by zooming out/in while dollying in/out to keep the size of the cube the same. The depth of the focus point can be changed by sliding a focal plane through the scene.

2 Related work

For mouse-based systems, camera control paradigms fall roughly into two categories, camera-centric and object-centric. In the camera-centric paradigm, operations are applied to the camera as if it were a real object in the scene. This mirrors camera placement in the real world, and many of the camera operations (dolly, pan, and roll) reflect that. The external parameters, position and orientation, can be specified either "through the lens", or by manipulating a pictorial representation of the camera in a second window. The internal camera parameters, with the exception of focal length, are changed through textual input.

In the object-centric paradigm, the camera is centered on an object and the viewpoint is rotated relative to the object (as if there were a virtual trackball around the object [Hultquist 1990; Henriksen et al.]). The camera can also be zoomed in and out. This paradigm is useful when there is a single object in the scene (or one object of importance) and the user is simply choosing a direction from which to view it.

Three or six degrees of freedom devices permit other interesting navigation techniques [Bowman et al. 1997], such as the palm-top world [Stoakley et al. 1995], the "grab and pull" approach [Poupyrev et al. 1996] and virtual fly-throughs [Wloka and Greenfield 1995]. The latter can also be used in mouse or keyboard-based systems if the camera's movement is restricted to a well-defined floor plane (most first-person shooters use this approach).

An alternative approach to directly specifying the camera is to use image-space constraints [Blinn 1988; Gleicher and Witkin 1992]. In this approach, points in the scene are constrained to appear at particular locations, or to move in a specified direction, and the system solves for the camera parameters that meet those constraints.

The recently-introduced IBar [Singh et al. 2004] is, in some sense, a specialization of the constraint approach, where the points are the points of the edge of a cube. Like the widgets presented here, the IBar is a screen-space widget where changing the widget changes one or two camera parameters. The IBar, however, maps *all* of the camera parameters to a single widget, which can be very confusing to a novice user.

In addition to being simpler, our technique also allows the user to pick any arbitrary rotation center in the scene, even one that is not tied to the surface or center of an object. Similarly, we provide a method for visualizing what part of the scene will remain fixed during a perspective distortion manipulation (Section 3.1).

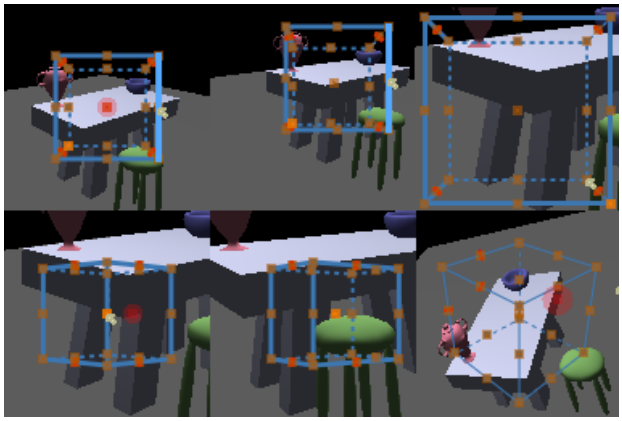


Figure 4: **User's View of the Cube widget:** The widget is initially viewed in a one-point perspective view. A camera pan and zoom is performed before switching to a two-point perspective view. The center of projection is changed in this view. Finally in a three-point perspective view, the camera is rotated about the focus point.

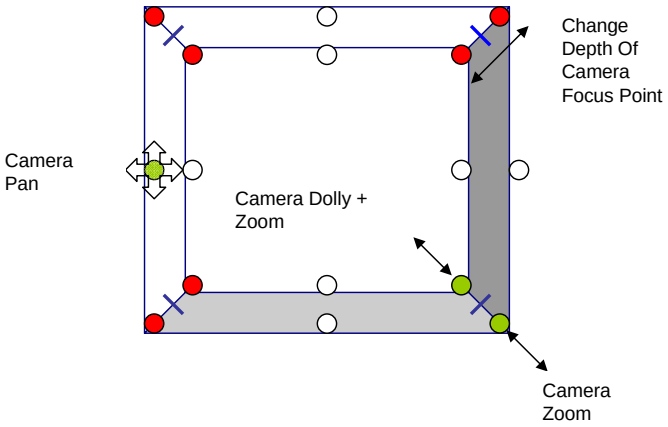


Figure 5: **1-Point Perspective Widget:** The corners of the widget's inner face are used for changing the perspective distortion while the corners of the outer face are used for camera zoom. The mid-points of the widget sides are used for panning the camera. The slider along the receding edges is used to change the depth of the focal plane. The Gesture Spot in the center of the cube is used to move between different perspective views. If the spot is dragged toward the right the view switches to a 2-point perspective. If it is dragged towards the left the view switches to the 3-point perspective.

3 User's view

The Cube Widget is a cube rendered in 1-point, 2-point, or 3-point perspective (Figure 2). It is either centered in the scene or attached to an object in the scene. Each part of the Cube Widget performs a specific camera operation. To avoid burdening the user with remembering all of the functions, we provide visual feedback indicating the changes that could be made to the current camera. As the user hovers over any part of the widget, an animation of the widget and scene changing in correspondence to that handle change is shown.

In order to move between the different types of perspective views mouse gestures are used. The user can grab and pull a part of the cube to switch to a different perspective view.

3.1 Perspective distortion

The cube widget can be used to control perspective distortion in the scene. Perspective distortion is function of the distance of the camera eye point to the object in question (dolly in). Unfortunately, changing the camera distance also changes the size of the object in the scene. To counter act this, we need to simultaneously change the camera zoom to keep the object the same size [Singh et al. 2004; Grimm et al. 2004]. This allows us to keep objects at a specific distance *d* along the Look vector the same size in the image plane. To visualize this distance, we draw a plane at the given depth. The position of this plane can be changed by moving a slider along the receding edges of the cube (Figure 3).

We extend this idea to the 2-point perspective rendering to control which objects will stay in the same place when changing the center of projection. In 3-point perspective, we use three focal planes, which together let the user specify an arbitrary rotation point in the scene.

Figure 4 shows construction of a camera view using the cube widget. The first five frames shows the manipulating the cube widget in 1-point and 2-point perspective views.

3.2 1-Point Perspective

The 1-point perspective view allows for camera zooming, panning, and perspective manipulation. The operation that is performed depends on the part of the widget which has been selected. Figure 5 shows the structure of the 1-point perspective Widget. In perspective manipulation mode, the front plane of the cube represents the fixed size plane.

3.3 2-Point Perspective:

The 2-point perspective view allows for camera zooming, dolly in, perspective distortion, panning, and changing the center of projection. When the center of projection is changed, the camera must be panned in the opposite direction to keep the scene from moving [Singh et al. 2004]. The focal plane determines what depth will be stationary.

3.4 3-Point Perspective:

Figure 7 shows the structure of the 3-point perspective widget. Using the 3-point perspective the user can rotate and pan the camera. The point about which the camera is rotated is the intersection of 3 focal planes aligned along the axes of the cube. Sliding these planes along their respective axes will change the point about which rotation takes place. The user can snap the widget to either the center or a point on the object, then adjust the rotation center.

The specified rotation can be either a virtual trackball [Henriksen et al.] or the standard 3-axis rotation. The virtual trackball is positioned around the cube, with the cube vertices on the sphere.

4 Visual Aids

Visual aids ease novice users into using the Cube Widget without the burden of remembering the functionality of a large number of handles. Our visual aids are similar to the idea of tool tips, except that we use animations of the widget and the scene to visually cue what a particular mouse action will do.

The animation is shown as a sequence of "ghosted" images taking the widget from its current state to the new one (see Figure 1).

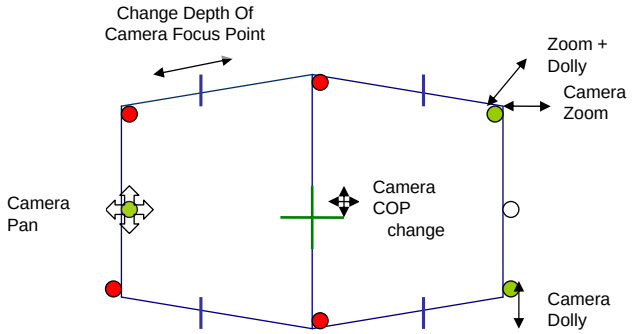


Figure 6: **2-Point Perspective Widget:** The center of the widget is used for center of projection changes. The corners can be used for either camera zoom only, camera dolly only, or both camera zoom and camera dolly. The slider is used to change the fixed translation focal plane. The mid-points of the sides are used to pan the camera. The Gesture Spot is used to switch to 1-point (left) and 3-point (right) perspective views.

Depending on the action being demonstrated, the scene change may be ghosted along with the widget (with the scene rendered using the changing camera).

Ghosting animations are used to indicate the effect of changing a handle, a slider, or "spinning" the widget using the Gesture Icon. For each handle we choose a canonical mouse movement direction and use that to generate a sequence of camera changes. The cursor is animated along with the scene and the widget. The Gesture Icon animation simply animates the cube widget itself, again with the cursor to indicate the mouse direction.

4.1 Implementation

The cube widget animations can be generated on the fly with the widget simply drawn on top of the scene. For the scene animations, we render the changed scene into the back buffer and then copy the image into a texture map. This texture is then alpha-blended on top of the rendering of the current scene. The texture images need not be at the same resolution as the full image; in fact, lower resolution images make it easier to visually distinguish the original scene from the ghosted one.

5 Bookmarks

It is very useful to be able to save cameras and snap back to them at will. For example, when modeling a surface, a user might bookmark a handful of orthogonal views and close-ups of complex geometry. An animator might also use bookmarks to start laying out an animation sequence. In both of these cases, we need to provide the user with a method for quickly searching through existing cameras. Although the user could simply create a text list, naming each camera, we believe that a visual search mechanism is more useful.

In our system we display bookmarks as icons that contain a simplified image of the scene. The icons contain visual cues about the relative position of the bookmarked camera with respect to to the current camera. The bookmark icons are placed around the boundary of the scene (see Figure 10). We employ two placement algorithms, which the user can switch between. In the first placement

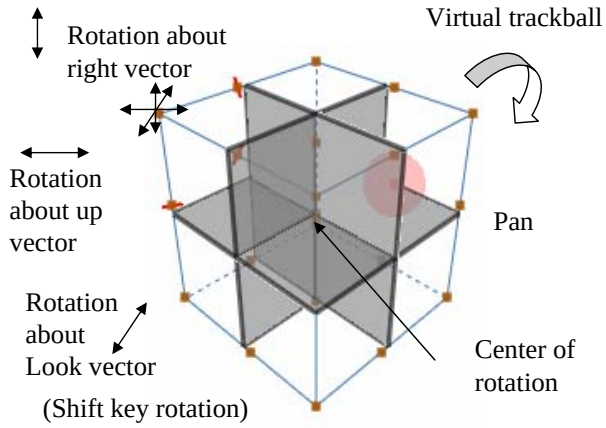


Figure 7: **3-Point Perspective Widget:** The midpoints of the sides of the widget are used for panning the camera. The corners change the orientation.

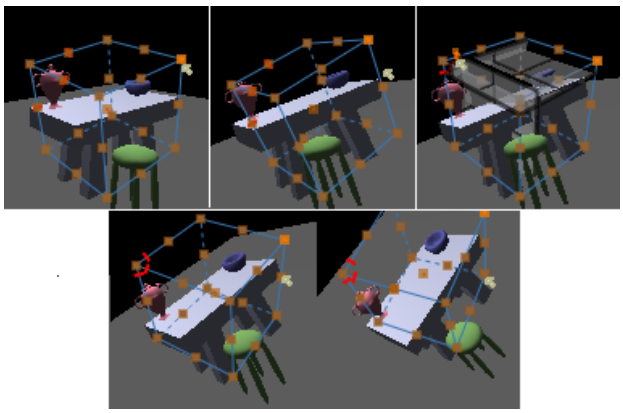


Figure 8: **Changing Point Of Rotation in 3-point Perspective:**The second frame shows the camera being rotated about the center of the cube. To change the center of rotation the focal planes are slid along the axes of the cube. The final frame shows rotation of the camera about the new center of rotation.

mode, the user simply places the icon in the boundary. In the second placement mode we automatically determine the 2D location based on the relative relationship of the bookmarked camera to the current camera. This second mode can be used to explore a scene by navigating through "nearby" bookmarks.

5.1 Implementation

To make the bookmark icons easier to see, we render the scene without textures, under a bright diffuse light, and highlight the silhouette edges [Gooch et al. 1998]. As the user mouses over the bookmark, the bookmarked camera's image is ghosted on top of the current camera.

To avoid continually consuming substantial screen real estate to display the bookmarks, we implement a re-sizable border. As the user moves into the border it re-sizes to display the icons, then shrinks again as the mouse moves back into the screen center, with a dead-zone in-between to prevent excessive border re-sizing.

Automatic placement: The location of the bookmark in the boundary indicates the relative position of the bookmark's eye point

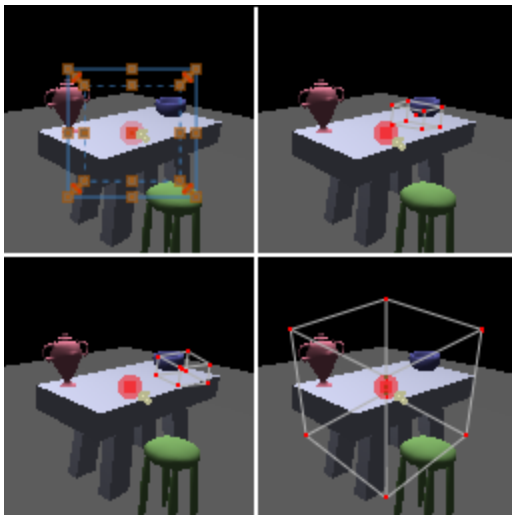


Figure 9: **Ghost Gestures:** As the user hovers over part of a Gesture Icon, the future perspective view of the cube is shown as an animation.

with respect to the current camera. This location is found by projecting the eye point onto the planes defining the view frustum. Points behind the current eye point are projected onto the inverse view frustum, then their 2D positions are inverted (objects behind the camera show up upside-down in the image plane).

The size and outline of the bookmark provide additional information about the distance of the eye point to the current optical axis and the relative orientation of the two cameras. We define a minimum and maximum size for the icons. We then find the minimum and maximum projection distances for all of the bookmarks and map the icon sizes appropriately.

The left and right outlines of the icon are scaled by l and colored c by the orientation of the bookmark's Look L_b vector with respect to the current camera's Look L vector:

$$\begin{aligned} t &= \left((< L, L_b > +1.0)/2\right)^3 \\ c &= (1,0,0)t + (0,0,1)(1-t) \\ l &= t l_{max} + (1-t) l_{min} \end{aligned}$$

where l_{min}, l_{max} are the minimum and maximum outline widths. Similarly the top and bottom outlines are determined by the relative orientation of the two camera's Up vectors. If the bookmark's eye point is behind the viewer, then we set the color to grey.

6 Implementation

Details of simultaneously changing the camera parameters for perspective distortion can be found in [Grimm et al. 2004] . Let d be the focus distance. The cube is draw at distance d along the look vector. The orientation of the cube is a composition of the inverse rotation of the camera matrix and the rotation matrix of that perspective. The cube is scaled by $d \tan\theta/2$, where θ is the zoom angle. In 1-point and 2-point perspective we offset the cube along the look vector by s to place the front of the cube at the fixed focus distance. To center a camera around an object, the calculation is the same except we place the cube at the object point and we scale the cube to match the object scale.

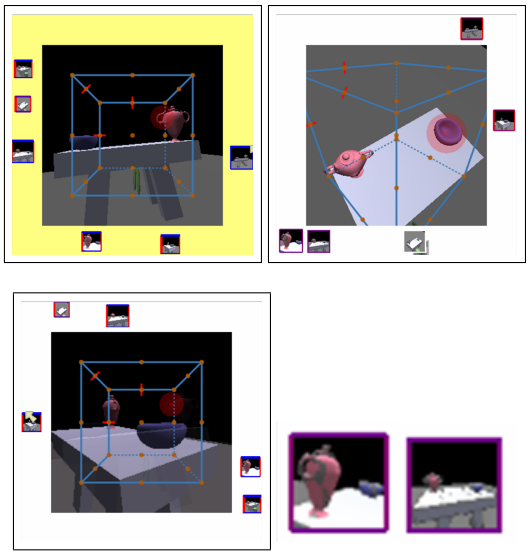


Figure 10: **Bookmarking:** A camera can be "bookmarked" at any point in time. The upper left figure shows an existing set of bookmarks placed by the user along the edge of the screen. The other two images demonstrate automatic placement. The lower right image is blow up of two of the icon images.

7 Conclusion

We have presented a camera manipulation widget that provides in screen visual feedback to the user regarding the state of the current camera projection. This widget allows the novice user to explore the full space of camera projection. Each perspective rendering presents the user with a subset of the camera parameters that can be easily explored. Visualization of bookmarks in the image plane extends the image space metaphor to multiple view points.

References

BLINN, J. 1988. Where am i? what am i looking at? In *IEEE Computer Graphics and Applications*, vol. 22, 179–188.

BOWMAN, D. A., KOLLER, D., AND HODGES, L. F. 1997. Travel in immersive virtual environments: An evaluation of viewpoint motion control techniques. *IEEE Proceedings of VRAIS'97*, 7, 45–52.

CARLBOM, I., AND PACIOREK, J. 1978. Planar geometric projections and viewing transformations. In *ACM Computing Surveys (CSUR)*, vol. 10.

CHEN, M., MOUNTFORD, S. J., AND SELLEN, A. 1988. A study in interactive 3d rotation using 2d input devices. In *Siggraph*, vol. 22, 121–130. Proc. of Siggraph '88.

COLE, R. V. 1976. *Perspective for Artists.* Dover Publications.

FOLEY, J., VAN DAM, A., FEINER, S., AND HUGHES, J. 1990. *Computer Graphics : Principles and Practice.* Addison Wesley.

GLEICHER, M., AND WITKIN, A. 1992. Through-the-lens camera control. In *Siggraph*, E. E. Catmull, Ed., vol. 26, 331–340. ISBN 0-201-51585-7. Held in Chicago, Illinois.

GOOCH, A., GOOCH, B., SHIRLEY, P., AND COHEN, E. 1998. A non-photorealistic lighting model for automatic technical illustration. *Computer Graphics 32*, Annual Conference Series, 447–452.

GRIMM, C., SINGH, K., AND SUDARSANAN, N. 2004. The ibar: A perspective-based camera widget. Tech. Rep. 7, Washington university in St. Louis.

HENRIKSEN, K., SPORRING, J., AND HORNBAEK, K. Virtual trackballs revisited. In *IEEE Transactions on Visualization and Computer Graphics*, vol. 10, 206–216.

HULTQUIST, J. 1990. A virtual trackball. In *Graphics Gems*. 462–463.

MICHENER, J. C., AND CARLBOM, I. B. 1980. Natural and efficient viewing parameters. In *Computer Graphics (Proceedings of SIGGRAPH 80)*, vol. 14, 238–245.

O'CONNOR JR., C., KIER, T., AND BURGHY, D. 1998. *Perspective Drawing and Application*. Prentice Hall.

POUPYREV, I., BILLINGHURST, M., WEGHORST, S., AND ICHIKAWA, T. 1996. The go-go interaction technique: Non-linear mapping for direct manipulation in VR. In *ACM Symposium on User Interface Software and Technology*, 79–80.

SINGH, K., GRIMM, C., AND SUDARSANAN, N. 2004. The ibar: A perspective-based camera widget. In *UIST*.

STOAKLEY, R., CONWAY, M. J., AND PAUSCH, R. 1995. Virtual reality on a WIM: Interactive worlds in miniature. In *Proceedings CHI'95*.

WLOKA, M. M., AND GREENFIELD, E. 1995. The virtual tricorder: A uniform interface for virtual reality. In *ACM Symposium on User Interface Software and Technology*, 39–40.

ZELEZNIK, R. C., HERNDON, K. P., AND HUGHES, J. F. 1996. Sketch: An interface for sketching 3d scenes. In *Siggraph*, 163–170.